Architectural Tradeoffs in Field-Programmable-Device-Based Computing Systems

Pak K. Chan*and Martine D.F. Schlag†

Computer Engineering
University of California, Santa Cruz
Santa Cruz, California 95064 USA

Reprogrammable Field-Programmable Gate Arrays (FPGAs) have enabled the realization of high-performance and affordable reconfigurable computing engines. We examine the architectural tradeoffs involved in designing general purpose FPGA-based computing systems with field-programmable gate arrays and field-programmable interconnects. The fact that FPGAs provide both programmable logic and programmable interconnects raises numerous design issues that need to be considered with care. Factors that influence the tradeoffs are routability, rearrangeability, and speed.

I FPGA-based computing systems

Reprogrammable FPGAs form the basis of high-performance and affordable reconfigurable computing, prototyping and emulation systems [1, 2, 3, 4, 5, 6, 7]. Interconnections schemes in these systems vary quite a bit. In SPLASH, PAM, QUICKTURN GANGLION and ANYBOARD [1, 3, 4, 5, 7], the FPGAs are directly connected to each other in a fixed pattern. In BORG and REALIZER [6, 2], logic and interconnection are separate. The REALIZER uses the *partial crossbar* interconnection architecture to connect the FPGAs, while the BORG has complete interconnection. In this paper, we shall investigate the design of interconnection architectures using multiple crossbars and discuss the architectural tradeoffs in the design of such FPGA-based computing systems.

We view FPGA-based computing systems as consisting of reprogrammable FPGAs, and reprogrammable interconnects. A number of commercial devices can serve as field-programmable interconnects:

Aptix FPICs have a place and route architecture which is not a crossbar [8]. Routing delay is not entirely controllable and predictable as a result of its place-and-route architecture. Each FPIC chip has a maximum of 940 user programmable I/Os.

IQ160 Each FPID chip has a maximum of 160 user programmable I/Os. The heart of the IQ160 is a 176 × 176 crossbar. Every port can be configured to connect to any port. Routing delay is entirely controllable and predictable as a result of this crossbar architecture [9].

TI crossbar The Texas Instrument SN74ACT8841 is a 16 × 16 crossbar with 4-bit word length. The crossbar is organized as 16 ports of 4-bit each. Routing delay is entirely controllable and predictable as a result of this crossbar architecture. Each crossbar has a maximum of 128 user programmable I/Os.

FPGAs have a place and route architecture. Routing delay is not entirely controllable and predictable as a result of this place-and-route architecture [10]. Each FPGA typically has a maximum of 200 user programmable I/Os.

Function-wise, the FPGAs and Aptix chips can be conceived of as devices capable of performing as crossbars under many circumstances (ignoring their unpredictable delays). This will be the view of this paper and we shall use *switches* and *crossbars* interchangeably in our discussion.

While all of the devices mentioned above can serve as programmable interconnects to provide flexible interconnection between computing elements in an FPGA-based computing system, the number of interconnections required may easily exceed the number of user programmable I/Os on a single device. We need to consider building a larger interconnection structure from these basic device packages which will be viewed as small crossbars. In this regard, the problem of building large crossbars from smaller crossbars is well studied in the context of telephone switching networks. Clos and Beneš networks are representative of rearrangeable networks made up of small crossbars. We shall review the basic terminology, concepts and known results related to Clos networks in Section II.

The notion of *folded-Clos networks* is introduced to facilitate the application of known results and a control algorithm to configure the crossbars in an FPGA-based system consisting of multiple small crossbars. This is one of the contributions of this paper. Then, we show that the fact that FPGAs have dual personality (both as programmable logic and programmable interconnects) which can be exploited to enable system tradeoffs in the design of an FPGA-based system using both FPGAs and field-programmable interconnect. The issues of speed, rearrangeability and routability are considered in Section IV.

*Supported in part by NSF Grant MIP-9196276 & MICRO
†Supported in part by NSF PYI Grant MIP-8896276 & MICRO

0-8186-3890-7/93 $03.00 © 1993 IEEE

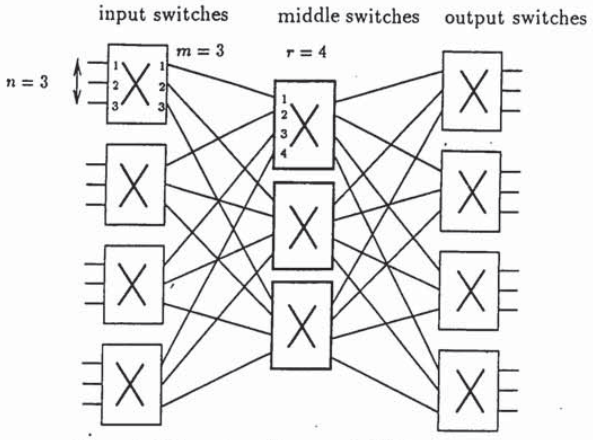

Figure 1: A three-stage Clos network $C(3,3,4)$.

II Rearrangeable CLOS switching networks

We review some basic terminology and concepts related to Clos networks in this section.

A connecting network is an arrangement of switches and interconnects allowing a set of input terminals to be connected to a set of output terminals in various combinations [11, 12]. All connections are point-to-point connections. An *assignment* for a given network is a list of input/output pairs which are to be connected; each terminal appears in at most one pair. An assignment is *realizable* if there exist disjoint paths in the network connecting all pairs of input/output terminals in the assignment. A network is *rearrangeable* if any assignment is realizable. *Nonblocking* networks have the property that in addition to being rearrangeable, any connection can be established between any idle pair of input/output terminals without rearrangements.

Three-stage symmetrical Clos network consists of two symmetrical outer stages of r $n \times m$ switches and an middle stage of m $r \times r$ switches. The network is completely characterized by three parameters n, m and r, or $C(n,m,r)$ collectively. A $C(3,3,4)$ Clos network is shown in Figure 1. The Slepian-Duguid theorem states that a 3-stage Clos network $C(n,m,r)$ is *rearrangeable* if and only if $m \geq n$ [11, 13]. For our purpose, we assume $n = m$, hence all switches are square and our Clos networks are rearrangeable.

Clos shows that a 3-stage Clos network $C(n,m,r)$ is *nonblocking in the strict sense* if $m \geq 2n - 1$ [14]. Clos also illustrates that it is possible to apply the principle recursively to form multi-stage rearrangeable or nonblocking networks. Clos assumes that all connections are point-to-point connections. In Section V, we shall consider an extension to multi-pin connections.

Although rearrangeable networks generally use fewer switches, they require careful and sometimes hard routing control algorithms. On the other hand, non-blocking networks require less demanding or no control algorithm [15]. The next section presents a known control algorithm for rearrangeable Clos networks based on bipartite matching [12].

II-A A control algorithm for rearrangeable Clos networks

Given an assignment A, a bipartite graph $G(A) = (V_1, V_2, E)$ can be constructed, where the vertex set V_1 consists of the input switches, and the vertex set V_2 consists of the output switches. There is an edge between switch $S \in V_1$ and $T \in V_2$ for each pair of input/output terminals in the assignment where the input terminal belongs to S and the output terminal belongs to T. The edges of this bipartite graph can always be partitioned into m disjoint matchings, and then each matching provides the at most r pairs of input/output terminals which can be realized by a single $r \times r$ switch in the middle stage of the three-stage Clos network. Thus, the problem of setting up the switches in a three-stage rearrangeable Clos network $C(n,m,r)$ can be formulated as edge-coloring with m colors (or equivalently partitioning the edges into m matchings) for the bipartite graph $G(A)$ where edges incident to the same vertex must be of distinct colors [12]. This problem can be solved in polynomial time [16].

III Architecture of FPGA-based systems

With the basics defined in the previous sections, we are now in a position to discuss FPGA-based systems consisting of the following components:

FPGAs: which can serve as computing elements, programmable interconnect or both. Figure 2

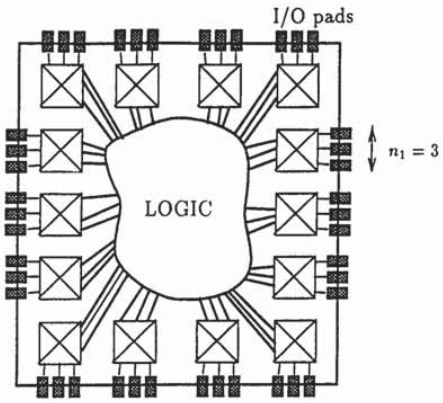

Figure 2: An FPGA modeled as consisting of programmable logic and switches connecting the nets to I/O pads.

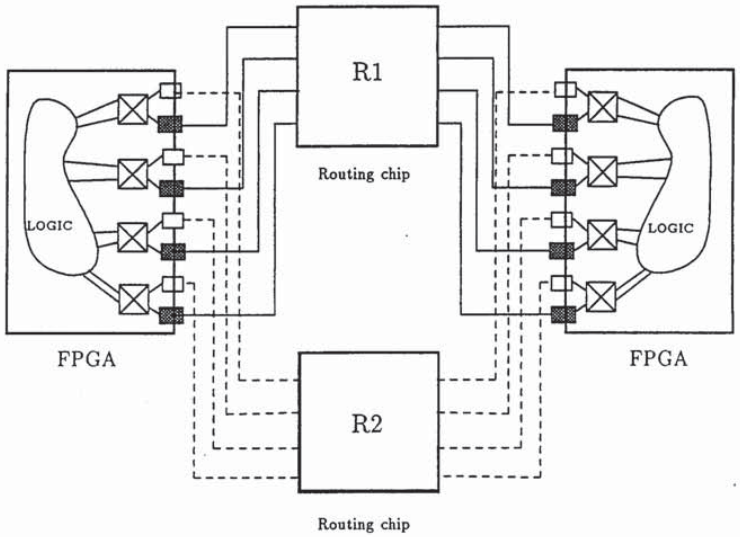

Figure 3: The BORG prototyping board viewed as as a three-stage $C(2, 2, 4)$ Clos rearrangeable network.

illustrates the last case in which an FPGA is viewed as a device consisting of programmable logic and a number of $n_1 \times n_1$ "switches" connecting the logic to the programmable I/O pads. The $n_1 \times n_1$ "switches" are realized by rerouting or reassigning the nets among the n_1 pads. The size of these switches (the amount of routing resources used to implement them) affects the routability of the FPGAs and hence the amount of logic that can be implemented on the device.

The software implication of this model is that we do not lock the pins to conform with board-level constraints. This is advantageous since

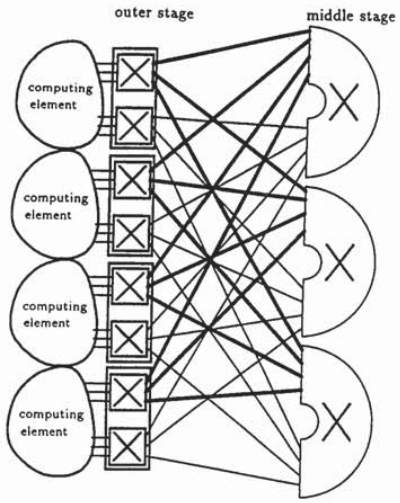

Figure 4: A folded-Clos network with $n = m = 3$, and $r = 4$.

reassigning or locking pins generally affects the routability of the FPGAs [10]. Instead, we allow the automatic placement and routing tool to choose freely the pin assignment. Afterward, one more routing step is required to use the "switches" to reroute the nets to conform with the board-level constraints.

Memory elements serve as local memories to the computing elements.

Host interface serves to communicate the FPGA-based computing system with the host machine to which the FPGA-based system is attached.

Programmable routing whose sole purpose is to provide programmable interconnect amongst the computing elements and/or memory elements. We assume that the number of interconnections required exceeds the number of user programmable I/Os on a single programmable routing device. The major issue is then the problem of building a large switch from the smaller switches.

We shall defer until Section VI the discussion of the impact of introducing other devices such as memory elements and the host interface to the system. So until then, we assume that there are only FPGAs and programmable interconnect in the system. As an example, consider the small programmable prototyping board BORG [6] which has two routing chips; it can be viewed as a three-stage *rearrangeable* Clos network. The FPGAs serve the dual purpose of programmable logic and programmable interconnection. The switches inside the FPGAs, provided by interchanging the nets among the programmable I/O pads, form the outer stages of the Clos network. The middle stage is realized by the FPGAs R1 and R2 which serve as routing chips, as illustrated in Figure 3. The fact that the I/O pads of the FPGA are connected to the routing chips in an alternating fashion makes the system depicted in Figure 3 a three-stage rearrangeable Clos network with $n = m = 2$ and $r = 4$. (The actual BORG uses Xilinx XC3030s as routing chips resulting in a $C(2, 2, 27)$ Clos rearrangeable network.)

The scheme shown in Figure 3 only works for $m = 2$. In next section, we shall discuss how to use Clos networks when there are more than two FPGAs, for $m > 2$.

III-A Folded-Clos networks

The Clos network and control algorithm outlined in Section II has a definite notion of input and output terminals. Therefore, in order to apply the control algorithm discussed in Section II one must identify the two terminals of a net either as input or output (irrespective of signal flow). This notion does not necessarily exist in an FPGA-based system, in which the computing elements have pads which can be reconfigured to be either input, output or bidirectional pads. The numbers and distribution of inputs/outputs change from application to application.

In this case it is most convenient to describe the *programmable-routing* component of an FPGA-based computing system as a folded-Clos network, as depicted in Figure 4. A three-stage folded-Clos network resembles a three-stage Clos network, but there is no distinction between input and output

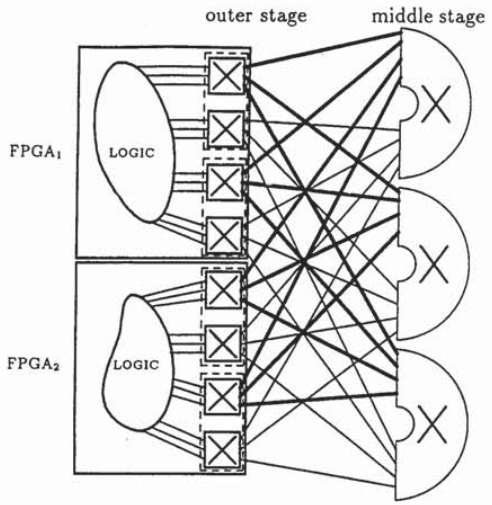

Figure 5: A folded-Clos network using the FPGAs as both computing element and programmable interconnect.

stages. We simply call the stage which connects directly to the computing elements the "outer" stage; it consists of r pairs of $m \times m$ switches. The "middle" stage consists of m $r \times r$ crossbars.

III-B. A control algorithm for folded-Clos networks

The control algorithm for the folded-Clos network depicted in Figure 4 can be adapted from the control algorithm for the three-stage Clos network depicted in Figure 1 as follows. First, directions are assigned to each of the nets so that each pair of $m \times m$ switches has m in-terminals and m out-terminals. To do this, we form a net-graph from the required connections, where each node corresponds to a group of $2m$ pins, and an edge is added for each net between the two nodes(groups) to which its two pins belong. We then traverse this net-graph visiting each edge once to partition the edges into edge-disjoint circuits (a node may appear more than once). We assign a direction to the edges as we traverse the net-graph. After the traversal, each node will have m in-edges and m out-edges.

Once the $2m$ edges have been divided, we can unfolded the foled-Clos network by separating the $2m$ terminals into two $m \times m$ switches and obtain a network identical to the three-stage Clos network. The Clos network control algorithm can be used to configure the $m \times m$ switches to realize any assignment. Notice that the direction assigned to an edge is purely for the purpose of using the edge-colorability control algorithm described in Section II-A, the direction is unrelated to the actual signal flow.

IV Architectural tradeoffs: FPGAs as both computing element and programmable interconnect

In this section, we examine the architectural tradeoffs involved in the organization of an FPGA-based computing systems. The issues are:

1. rearrangeability
2. routability, and
3. speed.

Any point-to-point connection realized by the architecture presented in Figure 4 has the delay of 3 switches. It is possible to trade routability with speed by exploiting the dual personality of FPGAs. Figure 5 shows a system resembling a folded-Clos network with the FPGAs serving as both programmable interconnect and computing elements. It is not difficult to show the network depicted in Figure 5 satisfies the Slepian-Duguid rearrangeability criterion (see Section II) providing the algorithm outlined in Section III-B has been used to identify the input and output stages. We eliminate two stages of switch delay in Figure 5 but risk the danger of having a non-rearrangeable network. This possibility is illustrated as follows. Assume that we control the utilization of the FPGAs such that they can be modeled as shown in Figure 2 with $m = 3$, $r = 4$.

Now, suppose that the utilizations of the FPGAs increase such that $n = m$ decreases from 3 to 2 while the external connections remain unchanged, as shown in Figure 6. Keep in mind that the interconnections between the outer stage and the middle stage are fixed. It is not hard to see that the network now is

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.